Fixed-Point Arithmetic Line Clipping

R. Mollá
Technical University of Valencia
Camino de Vera, s/n
Spain 46022, Valencia
rmolla@dsic.upv.es

P. Jorquera
Technical University of Valencia
Camino de Vera, s/n
Spain 46022, Valencia
pedjorhe@inf.upv.es

R. Vivó
Technical University of Valencia
Camino de Vera, s/n
Spain 46022, Valencia
rvivo@dsic.upv.es

ABSTRACT

This algorithm supports line clipping against a rectangular window with edges parallel to the screen. The algorithm combines different partial solutions given in the bibliography such as implicit area codes, decision trees, line ends comparison, symmetry or avoiding redundant calculations, mixing them with fixed-point arithmetic, explicit calculation reusing and dynamic monitoring. It may work in the fractional object space while still using integer arithmetic (fixed-point). It is faster than traditional algorithms. It provides more precision without using floating point arithmetic.

Keywords

Line clipping, fixed-point arithmetic.

1. INTRODUCTION

Many portable devices such as videogames consoles, mobile phones, PDAs, wearable computers, etc., have introduced graphics capabilities as an added value. These devices use very slow processors since low power consumption is mandatory. Neither graphic processor nor dedicated hardware is used, so efficient and simple software algorithms are mandatory.

This paper introduces a new straight line-clipping algorithm for 2D rectangular windows with edges parallel to the screen suitable for low cost integer CPUs without floating point units although it supports object space fractional numbers.

2. PREVIOUS WORKS

Many clipping algorithms are based on line parametric equations like Cyrus-Beck (CB) [Rog85] or the more efficient Liang-Barsky (LB) [Lia84]. Cohen-Sutherland (CS) [New79] approach uses another approach using explicit area codes to typify lines. This algorithm is more efficient since line ends are compared initially to every window edge.

WSCG POSTERS proceedings
WSCG'2003, February 3-7, 2003, Plzen, Czech Republic.

This process is able to trivially reject or accept a line even before doing any clipping calculation. The algorithms can be accelerated if they operate in the image space (integer arithmetic) instead of the object space (floating-point) [Dor90].

Using a convex window preprocess, the clipping cost may be almost constant [Ska93]. In practice, it can improve CB between 2.5 and 3 times. This algorithm may be improved getting down the computational cost from O(N) to O(lg N) [Ska94] [Bui99], where N is the amount of edges the clipping window has. If a good frame pre-process is performed, the clipping cost may get O(1) [Ska96]. Assuming that all lines are infinite, that is there are no segments, no line is trivially accepted simplifying the algorithm implementation [Ska99].

3. BACKGROUND

The FPC (Fixed Point Clipping) algorithm collects many good ideas made in the bibliography:

- It uses a decision tree to avoid typical clipping loop overheads [Nic87] [And91].
- Code reusing so that the line slope is calculated only once [Duv93].
- Early detection of trivial acceptation or rejection of segments [And91].
- No explicit area code such as the Cohen-Sutherland [New79] like algorithms. Thus, no codification process and no management are required.
- Successive comparisons take into account the ones just made before, avoiding calculation redundancy [Nic87] [And91] [Duv93].

- Code reduction by employing problem symmetry using vertical and horizontal reflection functions. [Nic87] [And91].
- Line ends are compared one each other before comparing them against the clipping window edges in order to reduce the amount of comparisons [Duv93].
- Precision increment is increased clipping always against original line ends instead of the intermediate clipped points [Dor90] [Duv93].

4. NEW METHODS

The algorithm also includes new ideas to improve efficiency not used in any previously known clipping algorithm [Mol01b]:

- Real time dynamic monitoring in order to detect the actual clipping load and improve trivial acceptation or rejection.
- Object space line clipping using 16/32 bits fixed-point arithmetic (12 millionths of a pixel resolution) without using floating-point arithmetic.
- Intermediate computations are made only when it is strictly necessary (line slope, line width or height). They are used during the clipping phase and by the line drawing algorithm if no rejection is made. Thus, the line drawing initialisation phase is reduced, speeding up the whole process.
- Implicit comparisons are explicitly carried out so that they are stored in intermediate variables in order to reuse them later if necessary.

Calculation reuse

Many times, doing a clipping process, typical statements like the following one come out

*if (X<Xmin) then Y0 = Y0 + (X - Xmin)*m;*

This statement may be written as

*if ((X - Xmin) < 0) then Y0 = Y0 + (X - Xmin)*m;*

This calculation redundancy can be avoided if the following change is included to obtain an implicit solution

*if ((Aux = (X - Xmin)) < 0) then Y0 = Y0 + Aux*m;*

Tests showed that typical statements are the fastest ones if the condition is false. But if the condition is true, then the typical statement is the slowest one. This solution presents an intermediate performance.

If the line is trivially rejected, its slope is never computed. If the line slope has to be calculated in order to perform a clipping, the slope is stored in order to reuse it in another clipping or when the line is drawn on the screen, reducing the line drawing algorithm initialisation phase by two subtractions and one division.

Fixed-point arithmetic

For the implementation of this algorithm, we used 32 bits fixed-point arithmetic. 16 bits were used for the integer part and 16 bits for the decimal part. The precision of this implementation was 12 millionth of a pixel. It is enough for today portable game players, printers o common CRT resolutions.

Monitoring

Line clipping is a process difficult to typify. Clipping rate depends on several factors: the clipping window size, its aspect ratio, segments size and position, ...

When using sequential algorithms, the cost of clipping against a window edge is added to the cost of clipping previously against other edges. Consequently, an implicit computational handicap appears for those cases detected at the end of the decision tree. Depending on the application clipping load, it is better to use a given tree or another. The main clipping challenge is to decide the main tree trunk to use in order to minimise the clipping cost.

A problem related to traditional algorithms is that they use rigid decision trees that cannot be altered during the execution phase. So, they do not need monitoring mechanisms since it is not possible to perform any change in the algorithm behaviour.

In this paper, we propose a scheme where a dynamic monitor checks and apply the corresponding decision tree that best matches the actual graphic load. This algorithm is selected by a planning procedure called at a frequency much lower than the sampling frequency.

According to process control theory, the monitoring algorithms have to take into account three aspects:

1. Significant parameters. In order to avoid very complex accounting, the FPC algorithm reduces monitoring to 3 possible cases: trivial rejection, trivial acceptation and clipping.
2. Sampling methods. Monitor overload has to be as light as possible in order to avoid monitoring affects the real load. Typically it is a reject/accept/clip accounting (variable unitary increment) that is reset every time the planning algorithm is called. Consequently, we have reduced the algorithm implementations to four possibilities: RxRyA, RyRxA, ARxRy and finally AryRx. For instance, RxRyA priorizes trivial rejection on X dimension, secondly trivial rejection on Y dimension and finally acceptation. We have not implemented, for simplicity, RxARy and RyARx.

3. Sampling frequency. There are several possibilities to determine when to call the planning algorithm or sample: every time a primitive is drawn, every object or every whole image. The longer the planning algorithm is called the lower the planning overhead is and the more sophisticated it may be.

If a segment has been clipped in a given way, the segments that belong to the same object will be "probably" clipped in the same way (spatial locality). Similarly, the clipping distribution of the i-th image will be quite similar to the i-th-1 and to the i-th+1 (temporal locality). This effect is emphasized the faster the graphic processor is. In this paper implementation, the sampling frequency is the drawing primitive. This sampling can be set permanently for all the screens or variable, depending on the application behaviour. The planning algorithm is called after a whole image accounting is made.

5. COMPUTATIONAL ANALYSIS

It is not easy to compare clipping algorithms since the clipping load depends on the window size, ratio, relative position or the segments to clip (clipping load). These features are closely related to the application type and user behaviour.

In order to compare different algorithms, we planned a laboratory test so that, under the same conditions, we could analyse real timing for all the algorithms.

Theoretical analysis

The LB algorithm uses lower comparisons than CS, but in general it performs more additions, products and divisions. In practice, LB is the slowest algorithm. CS algorithm is a bit faster. Nycholl-Lee-Nycholl (NLN) [Nic87] is the fastest one since it performs the lowest amount of instructions among all the traditional algorithms analysed. The FPC algorithm does more comparisons than the NLN, but it uses lower additions or subtractions. On the other hand, FPC always uses a lower or equal amount of products or divisions than NLN does. For this reason, in practice, the FPC algorithm is faster.

Empirical analysis

The theoretical computational cost showed that the FPC algorithm was the fastest but, it was not clear how the different monitoring possibilities could influence the clipping efficiency. On the other hand, it was important to verify empirically the theoretical studies. So, we tested all the algorithms in the laboratory using a synthetic load. It simulated a real clipping situation, equal for all of them. The tested algorithms were CS, a simplified CB version for rectangular windows and LB. The FPC algorithm was tested under the following monitoring conditions:

- Never monitoring and always giving priority to rejection. (FPC Never RA)
- Never monitoring and always giving priority to acceptation. (FPC Never AR)
- Continuous monitoring at the highest possible sampling frequency. (FPC Continuous)
- Constant sampling frequency but lower than the highest one. (FPC Constant)
- Adaptive sampling frequency depending on the change of sampled parameters. (FPC Adaptive)

The constant sampling frequency was fixed around every 5 screens and the adaptive frequency could change between 1 and 32 screens. It doubled if a change appeared. It changed to a half if no changes were observed during two consecutive plannings.

5.1.1 General results

The Table 1 shows the time spent by each algorithm. Results are in seconds. The known algorithms confirm the awaited results. So, the Cyrus-Beck algorithm is the worst one closely followed by Liang-Barsky. The best is Cohen-Sutherland. It doubles Liang-Barsky speed. The FPC average value reduces the best traditional computational cost down to 40%. That is, 250% speed up.

5.1.2 General comparison between different monitoring policies

Monitoring overload was a unitary increment after every line drawing. The planning algorithm was called after a whole screen drawing. In this sense, it can be said that the planning overhead computational cost is not worthwhile compared to the monitoring overhead. So, the different clipping algorithms analysis is finally reduced to determine which the best suitable monitoring frequency is.

The test worked with clipping windows never higher than 50% the total projection surface. For this reason, many of the lines were projected on average outside the clipping window. The tests were based on very homogeneous line distributions.

Those algorithm implementations that improve rejection instead of acceptation have a better result. If no monitoring is made, the result is even better. Thus, the algorithm FPC Never AR offers worse performance than FPC Never RA. The more frequent the monitoring is, the slower the algorithm performs. For this reason, the continuous monitoring offers worse timing than the constant one. Constant monitoring is worse than adaptive and this one is worse than those algorithms that never use monitoring. That is, $T_{NRA} < T_{NAR} < T_{Adap} < T_{Cte} < T_{Cont}$

Algorithm	Minimum	Maximum	Average
CS	4.53	7.3	5.47
CB	14.46	18.83	16.05
LB	8.83	18.17	12.32
FPC Never RA	0.98	5.82	2.37
FPC Never AR	1.175	4.28	2.48
FPC Continuous	1.32	5.21	2.65
FPC Constant	1.23	4.83	2.59
FPC Adaptive	1.2	4.78	2.57
Average FPC	1.18	4.98	2.53

Table 1. Clipping timing given by different FPC implementation and traditional algorithms

Using monitoring does not change the algorithm cost significantly. Differences are around 8% on average.

The FPC algorithm is based on a change of the clipping priority taking into account only the previous result. For this reason, sometimes, erroneous decisions may be taken, punishing performance.

6. CONCLUSIONS

Tests confirm conclusions seen in the bibliography respect to the algorithms CS, CB, LB and NLN. Theoretical analysis shows that the FPC is computationally more efficient than the other solutions. Empirical tests verify these results.

The FPC algorithm can clip lines with decimal ends apart from integer values. Fixed-point clipping is more precise than the integer one since no decimal part is lost during the floating to integer conversion (Clipping in object space vs. image space).

The FPC algorithm reuses calculations so that fixed-point arithmetic intermediate values can be used subsequently by line drawing algorithms like the stair algorithm [Mol01]. 32 bits fixed-point arithmetic has shown enough precision (12E-6 pixels) to manage properly the line projected onto the screen.

Dynamic monitoring has proven to be a good idea that can be exported to any other kind of algorithms in order to improve efficiency.

This algorithm can be upgraded to use SIMD instructions in parallel, increasing its speed even more.

7. ACKNOWLEDGMENTS

Our thanks to TIC-1999-0510-C02-01 project from Spanish Ministry of Science and Technology and PPI-UPV program for allowing us to develop this material.

8. REFERENCES

[And91] Andreev, R. D., Sofianska, E. New algorithm for two-dimensional line clipping, Computer & Graphics 15, No. 4, pp. 519-526, 1991

[Bui99] Bui, D. H. and Skala, V. New Fast Line Clipping Algorithm in E^2 With O(lgN) Complexity, International Conference SCCG'99, Budmerice, Slovakia, pp. 221-228, 1999

[Dor90] Dörr, M. A new approach to parametric line clipping, Computer & Graphics 14, No. 3/4 pp. 449-464, 1990

[Duv93] Duvanenko, V. J., Gyurcsik, R. S., Robbins, W. E. Simple and efficient 2D and 3D span clipping algorithms, Computer & Graphics 17, No. 1, pp. 39-54, 1993

[Lia84] Liang , Y-D. and Barsky , B. A. A new concept and method for line clipping, ACM Transactions Graphics 3, No. 1. pp. 1-22, 1984

[Mol01] Mollá, R. and Vivó, R. The stair algorithm, Journal Graphics Tools 6, No. 2. pp.17-25, 2001

[Mol01b] Mollá, R. Applications of the fixed-point arithmetic to the representation of low level graphical primitives, ISBN: 0-493-43368-6, 2001. Ph D. free full text in pdf format in wwwlib.umi.com/dissertations/fullcit/3030618

[New79] Newman, W. M. and Sproull, R. F. Principles of Interactive Computer Graphics, McGraw-Hill International Editions. 2nd Edition. ISBN 0-07-046338-7, 1979

[Nic87] Nicholl, Tina M., Lee, D.T. and Nicholl, R. A. An efficient new algorithm for 2-D line clipping: its development and analysis, Computer Graphics 21, No. 4,, pp. 253-262, 1987

[Rog85] Rogers, D. F. Procedural elements for computer graphics, McGraw-Hill, New York, 1985

[Sha92] Sharma, N. C. and Manohar, S. Line clipping revisited: two efficient algorithms based on simple geometric observations, Computer & Graphics 16, No. 1. pp. 51-54, 1992

[Ska93] Skala, V. An efficient algorithm for line clipping by convex polygon, Computer & Graphics 17, No. 4. pp. 417-421. 1993

[Ska94] Skala, Václav, O(lgN) Line Clipping Algorithm in E^2, Computer & Graphics 18, No. 4. pp. 517-524. 1994

[Ska96] Skala, V. Line Clipping in E^2 With O(1) Processing Complexity, Computer & Graphics 20, No. 4. pp. 523-530. 1996

[Ska99] Skala, V. and Bui, Duc Hui, Two New Algorithms for Line Clipping in E^2 and Their Comparison, Technical Report TR No. 108/99 University of West Bohemia, Plzen, Czech Republic